

Measuring Improvement when Using HUB Formats to Implement Floating-Point Systems under Round-to-Nearest

Javier Hormigo, and Julio Villalba, *Member, IEEE*

Abstract—This paper analyzes the benefits of using HUB formats to implement floating-point arithmetic under round-to-nearest mode from a quantitative point of view. Using HUB formats to represent numbers allows the removal of the rounding logic of arithmetic units, including sticky-bit computation. This is shown for floating-point adders, multipliers, and converters. Experimental analysis demonstrates that HUB formats and the corresponding arithmetic units maintain the same accuracy as conventional ones. On the other hand, the implementation of these units, based on basic architectures, shows that HUB formats simultaneously improve area, speed, and power consumption. Specifically, based on data obtained from the synthesis, a HUB single-precision adder is about 14% faster but consumes 38% less area and 26% less power than the conventional adder. Similarly, a HUB single-precision multiplier is 17% faster, uses 22% less area, and consumes slightly less power than conventional multiplier. At the same speed, the adder and multiplier achieve area and power reductions of up to 50% and 40%, respectively.

Index Terms—floating-point-arithmetic, digital-arithmetic, optimization, power-consumption, adders, multiplication

I. INTRODUCTION

THE rounding operation is performed in almost all arithmetic operations involving real numbers. There are several ways to perform this operation, although unbiased rounding-to-nearest has the best characteristics [1][2]. It provides the closest possible number to the original exact value, but if the exact value is exactly halfway between two numbers, then it is selected randomly. The most commonly used approach is the tie-to-even method, which is the default mode of the floating-point IEEE-754 standard (see [3]).

However, the implementation of this rounding mode is relatively complex, and the area and delay introduced for rounding circuits may be very large, since they normally lead in the critical path. For this reason, it is only generally used in floating-point (FP) circuits. Many researchers have proposed different architectures to reduce the impact of this delay by merging rounding with other operations or removing it from the critical path. For instance, an FP adder was proposed in [4], such that if the result of an addition is input to another one, the incrementation required for rounding up is postponed until the next operation. In [5], a dedicated circuit to compute the sticky bit in parallel with the main path was proposed with the aim of accelerating the implementation of multiplication.

The authors are members of the Department of Computer Architecture, Universidad de Málaga, Málaga E-29071 Spain (e-mail: fjhormigo@uma.es).

This work was supported in part by the Ministry of Education and Science of Spain under contracts TIN2013-42253-P.

A compound adder (a circuit which, having a carry-save input, delivers the results and the result plus one) was proposed in [6] to generate the rounded result of any operation. In [7], three different methods were compared for multipliers which simplify rounding decisions and merge the rounding up with the computation of the operation. Similarly, [8] proposed combining rounding with the final addition to convert the carry-save solution to conventional representation. In [9], a rounding scheme was presented for high-speed multipliers based on a rounding table and prediction.

A totally different approach would be to use a new real-number encoding, in order to simplify the implementation of round-to-nearest. Thus, the problem would change from optimizing the rounding operation to dealing with arithmetic operations under the new number representation. This proposal is found in [10] with Round-to-Nearest representations (RN-representations) and [11] with Half-Unit Biased (HUB) formats. Together with other advantages, these new formats allow performing round-to-nearest simply by truncation. On the other hand, these new formats are based on simple modifications of conventional formats and so could be applied to practically any conventional format. In this article, we focus on HUB FP formats.

The efficiency of using HUB formats for fixed-point representation has been demonstrated in [12] and [13]. By reducing bit-width while maintaining the same accuracy, the area cost and delay of FIR filter implementations has been dramatically reduced in [12], and similarly for the QR decomposition in [13].

In this article, we perform a quantitative estimation of the benefit obtained using HUB formats to implement FP computation systems under round-to-nearest. Some preliminary results for half-precision FP adders and multipliers were presented in [14]. This previous work shows that the area and power consumption of a basic FP adder could be improved by up to 70% for high frequencies when using HUB formats, whereas they remain the same for the basic FP multiplier. In addition to a deeper analysis, in this article we extend these results to other sizes and circuits, such as converters. In comparison to previous work, the main contributions of this article are:

- A detailed architecture for basic adder and multiplier to deal with HUB numbers
- A study of the conversions between different FP formats and the corresponding architectures
- The experimental comparison of accuracy between HUB

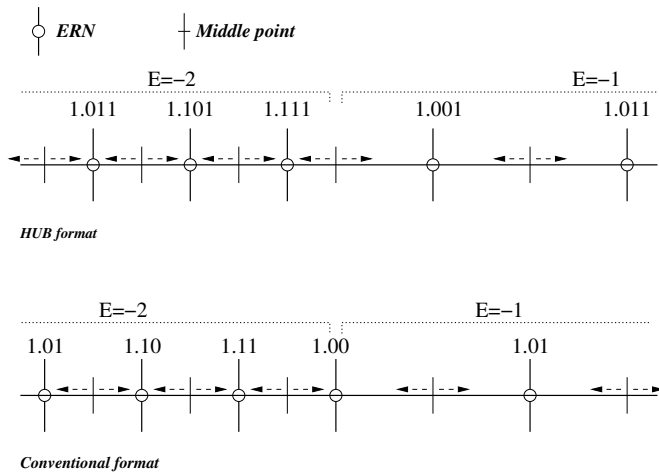


Fig. 1. Example of ERNs for a conventional FP format and its HUB version

and conventional formats

- Measures of improvements in area, speed, and power consumption for single- and double-precision adders, multipliers, and converters.

The rest of this paper is organized as follows: Section II reviews the fundamentals of HUB FP formats, focusing on rounding operations. Section III addresses the architectures for implementing FP operations under HUB formats, specifically addition, multiplication, and conversions. In Section IV, we provide an experimental error analysis to confirm the viability of using HUB formats instead of classic ones, and also compare the results of the ASIC implementation of a basic adder, a basic multiplier, and several converters. The conclusions are presented in Section V.

II. HALF-UNIT BIASED FP FORMATS

A HUB FP format should include a significand that is represented by using a HUB fixed-point number, and an exponent that is represented in any conventional way [11]. A HUB fixed-point format is produced when the Exactly Represented Numbers (ERNs) of a conventional representation are increased (or biased) by half Unit in the Last Place (ULP). This shifting of the ERNs could be seen as Implicit Least Significant Bit (ILSB) set one. For example, the HUB version of the IEEE-754 single precision has 25 bits for the significand, where the first and last bits are implicit and equal one, but only 23 bits are stored, as in the conventional version.

Fig. 1 shows an example for a binary FP format with 3-bit significand. Given a real value, its representation using either conventional or HUB formats will produce different rounding errors, although the accuracy of both format is the same [11]. In fact, the rounding errors for both formats are complementary (i.e., the addition of both rounding errors equals 0.5 ULP).

The main advantage of computing with HUB formats is that the two's complement operation is implemented simply by a bit-wise inversion and rounding-to-nearest by truncation. The unbiased rounding may require forcing the LSB to zero, when all discarded bits are zero [11].

III. BASIC OPERATIONS UNDER HUB FORMATS

The general procedure to operate with HUB numbers involves the following steps [11]: Firstly, the ILSB is explicitly appended to the significand of input operands. Secondly, the operation is performed in a classic way such that all bits of the significand result before rounding are obtained. Finally, the significand is rounded simply by truncation. However, since the ILSB is a constant value, the datapath could be further optimized depending on the specific operation. Next, we develop several architectures to support HUB numbers. These architectures are adapted from the basic architectures described in [1]. We are aware that many optimizations to these architectures have been proposed in the literature, such as those presented in [15][16][17][18][19][20][21][22][1][2][23]. However, none can be selected as the best, since this selection depends on many different factors. Even if the more relevant ones were selected, it would not be possible to review all of them in this article. Thus, we simply describe the adaptation of these basic architectures with the aim of their being used as examples to investigate the implementation of other much more sophisticated approaches.

Next, we study in detail the architectures to implement two basic operations, addition and multiplication, and different conversions. For this purpose, let us consider an m -bit significand including the leading one, which is explicitly represented to facilitate explanation. In contrast, the ILSB of the HUB version is not included in m . We focus on the implementation of the significand path, because the exponent path remains practically unchanged. We will draw attention to any modification required.

A. FP Addition for HUB numbers

A basic FP addition for conventional numbers requires several steps [1], which could be implemented using the significand data-path shown in Fig. 2(a). Firstly, the operand exponents ($d = E_x - E_y$) are compared and the significands aligned accordingly. The latter is usually performed by right shifting ($|d|$ bits) the significand corresponding to the number with the lowest exponent, which is selected using the swap module and the sign of d . The computation of the sticky bit corresponding to the bits shifted beyond the precision of the significand is also performed. The sticky bit is required for the computation of two's complement and rounding.

Secondly, either the effective addition or subtraction of the aligned significands is performed for the $m+3$ MSBs (the significand plus guard, round, and sticky bits). In general, in order to perform the subtraction, the significand corresponding to the lower exponent is previously one's complemented using the significand comparator and the conditional bit inverters. Moreover, the sticky bit is introduced in the adder to complete the two's complement transformation.

The result of the addition has to be normalized by shifting one bit to the right, if an overflow is produced. Otherwise, it is shifted to the left if there are leading zeros whose number is computed in the leading one detector (LOD). Besides normalization, the result has to be rounded based on the two LSBs of the result and the sticky bit. However, no roundup is

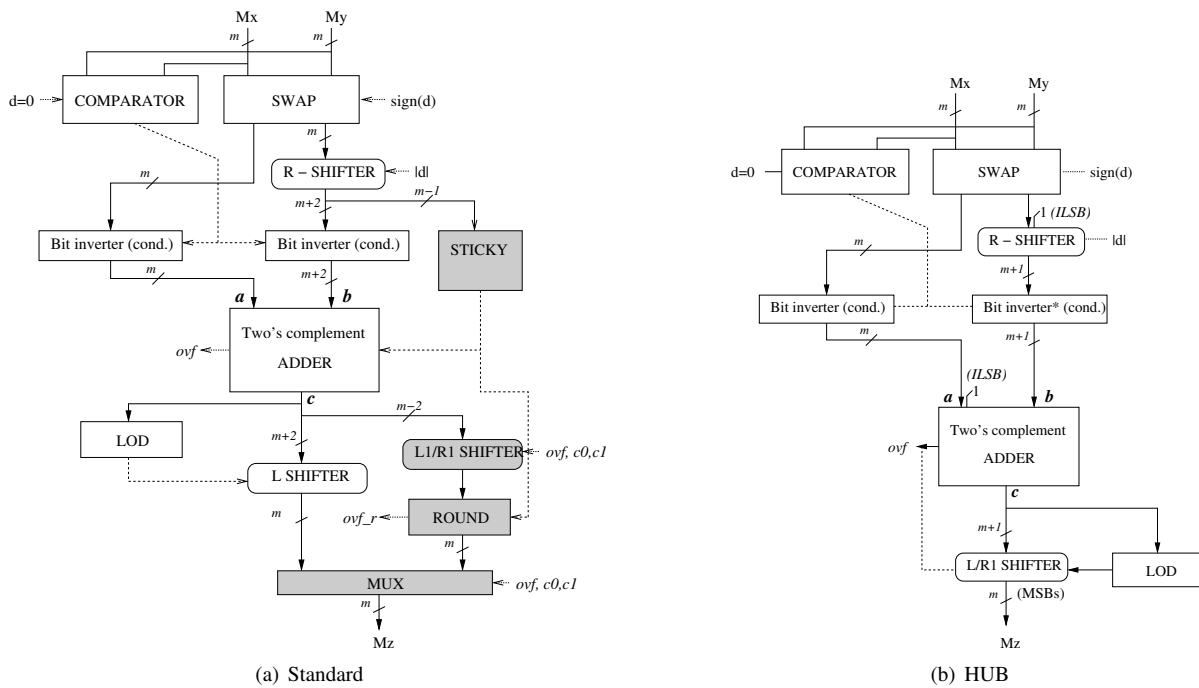


Fig. 2. Basic FP adder architectures for standard (a) and HUB numbers (b).

required when the result has at least two leading zeros [23][1]. Thus, left-shifting and rounding are performed in parallel paths. On the other hand, the new exponent is also generated in a parallel path. If the result of addition is rounded up, an overflow may be produced, which requires a new correction of the exponent.

The same basic architecture could be used for HUB numbers, although the significands are one bit larger and the rounding circuit is removed. However, knowing that the ILSB always equals one, the significand data-path is further optimized, as shown in Fig. 2(b). The first difference is in the right-shifter used to align the significands, because the ILSB has to be included at the input to obtain a correct result if no shifting is performed. Furthermore, the sticky bit computation logic has been removed. Given that the ILSBs of both significands equal one, the sticky bit is always one for non-aligned significands [11]. Moreover, the sticky bit is not required for aligned significands because shifting is not performed.

In this HUB architecture, we should note that the conditional bit inverters directly perform the two's complement, as explained in Section II, and no carry input is required in the fixed-point adder. The conditional inverter at the output of the right shifter has to be modified to control when shifting is not performed. In this case, the ILSB is explicitly represented by the LSB of the output. It then has to be set to one after the inversion to complete the two's complement operation (see Section II).

On the other hand, Fig. 2(b) shows that the ILSB of the second operand is appended at the corresponding input of the fixed-point adder. Despite this, the fixed-point adder is slightly shorter than the one shown in Fig. 2(a). The latter requires two guard bits and the carry input for the sticky bit

(i.e., $m+2$ bits) due to the rounding operation. However, in the HUB approach shown in Fig. 2(b), no guard bits are required because rounding is performed by truncation. Thus, the fixed-point adder has only $m+1$ bits (one additional bit to support the ILSB). In fact, the ILSB is shown in the architecture to simplify the explanation although, as presented in [11], this fixed-point addition can be implemented using an m -bit adder and an inverter.

Finally, the rounding path (gray in Fig. 2(a)) is removed, because rounding is simply performed by truncation. Consequently, given that explicit rounding up is not performed for the HUB architecture, overflow could not occur after rounding. Thus, the additional correction of the exponent required in Fig. 2(a) is eliminated, which also simplifies the exponent data-path.

B. FP multiplication for HUB numbers

A classic FP multiplication is simpler than FP addition [1]. The new exponent is computed by adding the exponents of the input operands, whereas the significands are multiplied thus obtaining a value that is double the size. The result of the multiplication is normalized by shifting it one bit to the right, if it is required. Finally, the resulting number is rounded to fit the size of the significand. The rounding requires computing the sticky of the $m-2$ LSBs of the result of the fixed-point multiplication, and the addition of one ULP for rounding it up. Fig. 3(a) shows the significand data-path of a straightforward implementation of the conventional FP multiplication.

When HUB numbers are used, the significands again have to be appended with the hidden ILSB that equals one, as shown in Fig. 3(b). Thus, the fixed-point multiplier is one bit larger, which increases the size of the fixed-point multiplier. In contrast, the computation of the sticky bit is again prevented [11].

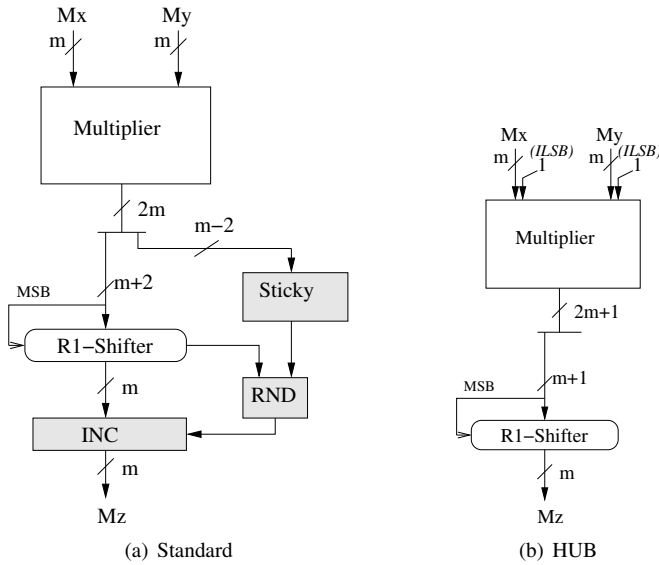


Fig. 3. Basic FP multiplier for standard (a) and HUB numbers (b).

To simplify the explanation, let us assume that the significand is scaled such that the ILSB is the only fractional bit. Let us call the integer part of both input significands A and B. The product of both significands is then

$$(A + 1/2) \cdot (B + 1/2) = A \cdot B + \frac{1}{2}(A + B) + 1/4 \quad (1)$$

Given the last addend of this result, we conclude that the LSB of the result of multiplication is always one. Thus, the sticky bit is always one and no logic is required to compute it. Moreover, given that the rounding is simply performed by truncation, the final incrementer is not required (see Fig. 3(b)). Therefore, as shown in Section IV-B, although the area of the fixed-point multiplier increases, the overall area decreases.

C. Conversion between FP numbers of different sizes for HUB numbers

Conversion between FP formats with different sizes is another operation that could benefit from using HUB formats. We will focus on the conversion of the significand, since the conversion of exponent is not affected when using HUB formats (because HUB FP formats use a conventional number to represent the exponent).

The conversion from an FP format to another format with a more narrow significand requires a rounding operation. For instance, the 53-bit significand of a double precision number has to be rounded to 24 bits to turn it into a single precision number. Fig. 4 shows an example of this situation. The standard round-to-nearest tie-to-even rounding mode would require computing the sticky bit, determining the need for rounding up, and incrementing the truncated significand value if required. These operations involve a considerable amount of hardware. Moreover, the rounding-up operation may produce a significand overflow, which would require incrementing the exponent by one. However, under HUB formats, all these operations are avoided, and a simple truncation of the significand will produce the correctly rounded conversion. We should note

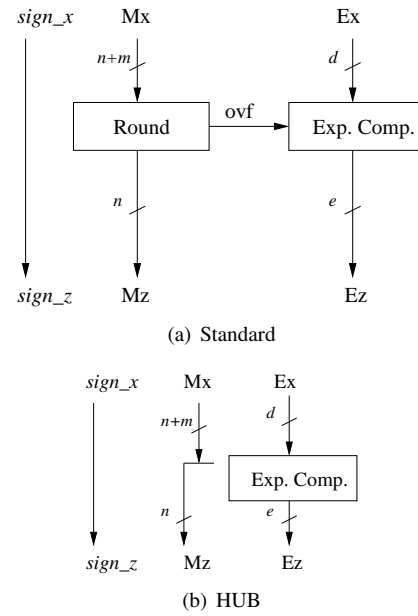


Fig. 4. Basic converter from an FP format to a narrower one for standard (a) and HUB numbers (b).

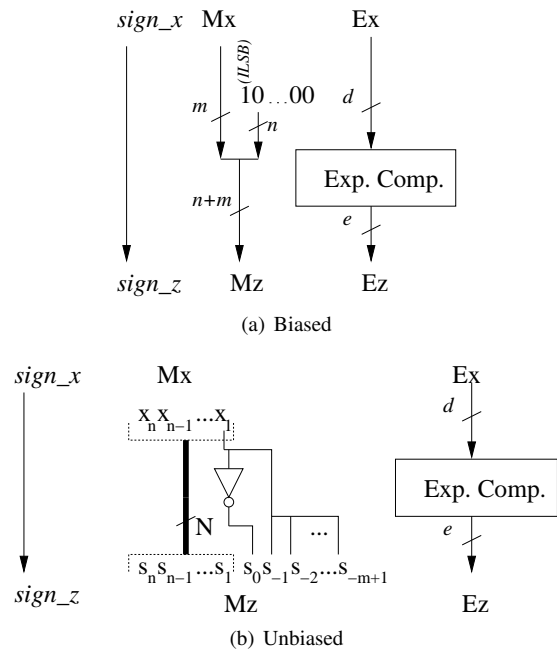


Fig. 5. Basic converter for an FP format to a larger one for HUB numbers.

that the ILSB of the original number guarantees that the sticky bit is always one, which means that the tie case does not exist.

Apart from the dramatic simplification of the conversion operation, the use of HUB formats prevents the double rounding error [11]. This well-known problem may occur when a number is rounded twice in a row (e.g. when the result of an arithmetic operation is first rounded to double precision, and then to single precision). Using standard numbers, this operation may lead to the final value having an error greater than 0.5 ULPs. However, this cannot occur when using HUB numbers[11].

On the other hand, conversion the other way round requires

expanding the significand to fit it into the new size. This is a trivial operation for standard format, because as many zeroes as needed are appended to the least significant part of the significand. In relation to HUB formats, a similar solution could be applied, although the MSB of the bits appended should be one (the ILSB) instead of zero. This approach is shown in Fig. 5(a).

Due to the new ILSB of the generated HUB number, the circuit of Fig. 5(a) produces a rounded tie-to-away significand because the significand is always rounded up. The error introduced due to this rounding is negligible, because the error associated with the original FP number is usually several order of magnitude greater. For example, the representation of the value 0.1 under then IEEE-754 single-precision standard produces an error that equals $-1.4901161 \times 10^{-9}$ assuming the default rounding mode. When it is extended, the same amount of error is transferred to the double precision number. Similarly, the error under the single-precision HUB format is 2.2351743×10^{-9} which, in this case, is reduced 6.94×10^{-18} by appending the ILSB of the double-precision HUB number. It can clearly be seen that this rounding-up operation barely affects the original error value.

Nevertheless, if this bias is still a problem for certain applications, another solution is shown in Fig. 5(b). This approach selects the direction of the rounding based on the explicit LSB of the input significand. If this LSB is zero, the significand is extended as in the previous solution; thus, it is rounded up. If the LSB is one, it is extended with a zero followed by ones, which actually produces a rounding down (recall the ILSB set to one of the input significand).

D. Conversion between conventional and HUB formats

The conversion between numbers under a conventional format and its corresponding HUB format (i.e., both formats having the same number of explicit bits) may be required when data are exchanged between systems working in these different formats. Given that those formats do not share any ERN (except special cases), this conversion always implies a rounding and, consequently, a rounding error. In addition, each ERN of one format is always at the midpoint between two ERNs of the other format, i.e., it is a tie case. This fact simplifies the hardware, because the computation of the sticky bit is not required; however, the magnitude of rounding error is always 0.5 ULPs.

The conversion from conventional FP format to HUB format could be performed without any explicit operation. In this case, given that the HUB format virtually appends the ILSB to the initial significand, an implicit rounding up of the magnitude is always produced. Thus, a tie-to-away is actually produced, because an effective rounding up is obtained for positive numbers, whereas it is a rounding down for negative numbers. However, if this tie-to-away behavior is not desired (for example, when the sign of the input numbers is not equally distributed), a kind of tie-to-even rounding could be easily implemented. To do this, the explicit LSB of the significand (the second LSB, if the ILSB is taken into account) is forced to zero. In this way, significands which were initially even are rounded up, whereas the odd ones are rounded down.

TABLE I
EXAMPLES OF CONVERSIONS BETWEEN THE SIGNIFICAND OF HUB AND CONVENTIONAL NUMBERS

conventional to HUB			
conventional	tie-to-away	tie-to-even	
1.011 (1.375)	1.011 (1.4375)	1.010 (1.3125)	
1.110 (1.75)	1.110 (1.8125)	1.110 (1.8125)	
-1.101 (-1.625)	-1.101 (-1.6875)	-1.100 (-1.5625)	
-1.010 (-1.250)	-1.010 (-1.3125)	-1.010 (-1.3125)	
HUB to conventional			
HUB	tie-to-zero	tie-to-even	tie-to-odd
1.001 (1.1875)	1.001 (1.125)	1.010 (1.250)	1.001 (1.125)
1.010 (1.3125)	1.010 (1.250)	1.010 (1.250)	1.011 (1.375)
-1.111 (-1.9375)	-1.111 (-1.875)	-1.000 (-2.000)	-1.111 (-1.875)
-1.110 (-1.8125)	-1.110 (-1.75)	-1.110 (-1.75)	-1.111 (-1.875)

On the other hand, the conversion from HUB FP format to conventional format could also be performed without any explicit operation. The ILSB of the significand is virtually removed and then the magnitude is implicitly rounded down. Consequently, doing nothing to convert the numbers actually produces a tie-to-zero rounding. In contrast, producing a tie-to-even rounding requires much more hardware because it involves incrementing the number if its LSB equals one (recall that it is known in advanced that it is a tie case, and thus no other testing is required). However, tie-to-odd behaves similarly to tie-to-even rounding mode and it is much easier to implement simply by setting the LSB of the number to one. In this way, even numbers are rounded down, whereas odd numbers are rounded up. Table I shows several examples of these conversions using a 4-bit significand for different rounding modes. The numbers between parentheses are the corresponding decimal values (recall that HUB numbers add 0.5 ULP to the corresponding conventional value).

IV. IMPLEMENTATION RESULTS AND COMPARISON

In this section, we experimentally study the convenience of using the proposed HUB FP formats to implement real-number computation. Firstly, we provide an experimental error analysis to confirm that the use of HUB formats does not damage the accuracy of the computation. Secondly, we analyze the main results of the hardware implementation of the proposed HUB FP circuits compared to the classic implementation.

A. Experimental error analysis

An empirical error study is provided to demonstrate that HUB formats could be used instead of the IEEE standard in FP-specific applications, while guaranteeing the same level of precision. In a first experiment, we tested the arithmetic operations. In these experiments, we utilized 32 bits two's complement fixed-point numbers within the range $(-1 \ 1)$ (i.e., one sign bit and 31 fractional bits) as exact real input numbers. They were converted into an internal 32-bit FP format with only 24 bits for the significand. Thus, a rounding was required for said conversion. Two different internal format were tested,

TABLE II
STATISTICAL PARAMETERS OF ROUNDING ERROR DISTRIBUTION.

Operation:	addition			
Parameters:	min	mean	max	σ
HUB	-7.404e-08	-5.1649e-12	7.404e-08	2.0998e-08
IEEE	-7.404e-08	2.8433e-12	7.404e-08	2.1346e-08
Operation	multiplication			
HUB	-8.3704e-08	1.7795e-11	8.2135e-08	1.3458e-08
IEEE	-8.3168e-08	-1.1068e-11	8.1511e-08	1.3449e-08
Operation	Double to single precision conversion			
HUB	-2.9802e-08	1.4921e-11	2.9802e-08	1.302e-08
IEEE	-2.9802e-08	-1.1596e-11	2.9802e-08	1.3008e-08

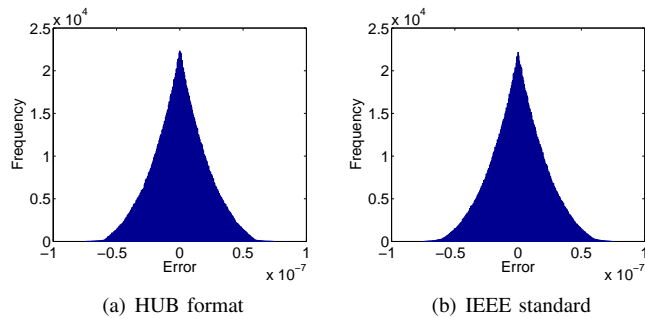


Fig. 6. Histogram of the rounding error for the addition experiment.

the standard IEEE-754 single precision and its corresponding HUB format.

To test the addition operation, two millions exact results corresponding to the addition of two input real numbers were calculated by using 32-bit fixed-point arithmetic (excluding the results which produced overflow). Moreover, the FP results of adding the same pairs of numbers that were previously converted to single precision FP format were computed for both the IEEE standard and HUB format. For the former, a standard CPU was used to perform the computation, whereas for the latter an FPGA implementation of the HUB adder presented in Section III-A was used. These results were converted back into fixed-point representation (considered exact in our experiment) and compared to the exact results obtained using fixed-point addition.

The calculated error comprises the error in the operation itself and in conversions. We should note that the values of the rounding error corresponding to both approaches are always different. This happens because the value used for representing the exact real number is different on each internal representation (see Section II). However, the probability distributions of these errors are quite similar, as shown by the histogram of the rounding error for both HUB format and IEEE standard shown in Fig.6. Moreover, Table II shows the main statistical parameters corresponding to these errors. It can be seen that these parameters are very similar when both representations are compared.

A similar experiment was run for multiplication. Again, two millions exact multiplication results were compared with the results obtained with FP internal representation when using the IEEE standard single precision and its equivalent

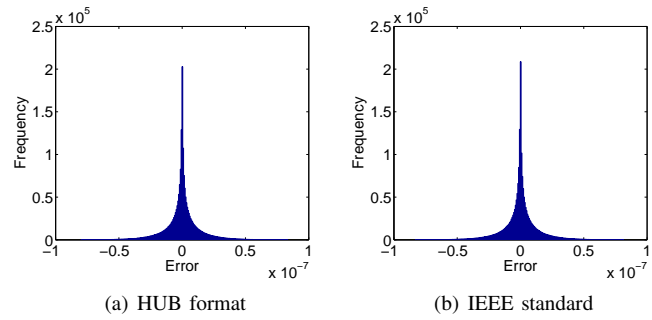


Fig. 7. Histogram of the rounding error for the multiplication experiment.

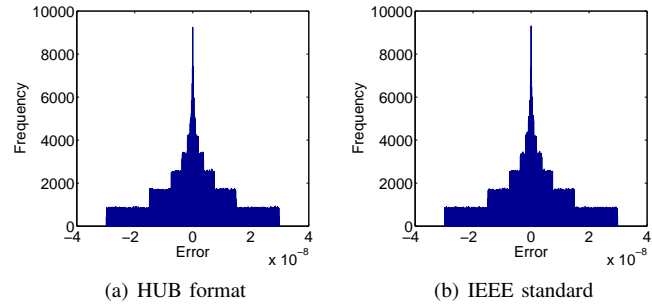


Fig. 8. Histogram of the rounding error for double- to single-precision conversion.

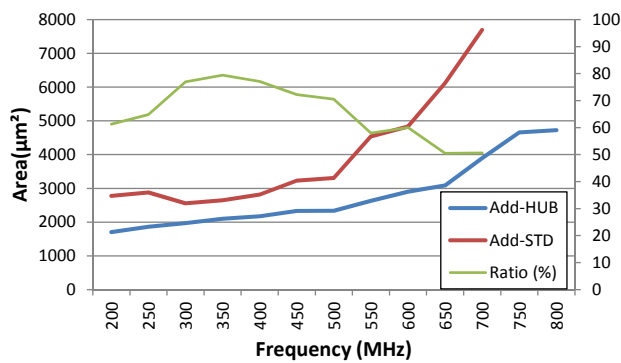
HUB format. The input operands were also 32-bit fixed-point numbers although, in this case, only positive numbers were used, whereas 64-bit fixed-point arithmetic was used to calculate the exact multiplication result. Similar to the addition experiment, although the values of the rounding error for both representations are different, their statistical parameters are very similar as shown in Table II and Fig. 7 by the histogram of the rounding error for both the HUB format and IEEE standard.

In another experiment, we measured the precision of the conversion from double to single precision. One million double-precision real numbers were randomly generated and converted into single-precision format. They were then converted back into double precision and compared with the original numbers. As expected, the statistical distributions of the error for both standard and HUB formats were once again very similar. Fig.8 shows the histograms of the rounding error for both approaches and their statistical parameters are shown in Table II.

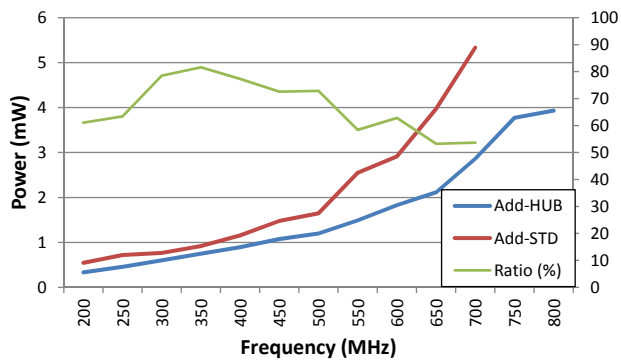
The theoretical and experimental results indicate that HUB formats could substitute for conventional FP representation to deal with real-number computation. Clearly, the use of the HUB format to solve real-number calculation would not provide exactly the same results as those obtained by use of the IEEE standard, although the accuracy of this calculation would be the same.

B. Implementation results

To measure the improvements obtained by using the HUB format and the corresponding circuits, the basic circuits studied in section III have been described in VHDL and implemented

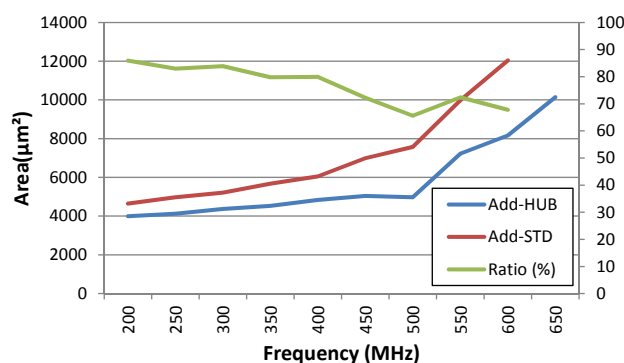


(a) Area

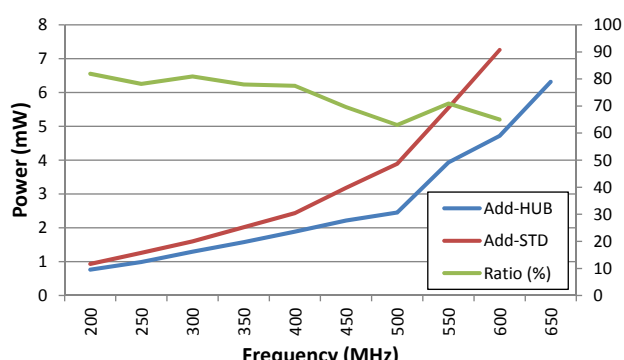


(b) Power

Fig. 9. Comparison of adder implementations for single precision



(a) Area



(b) Power

Fig. 10. Comparison of adder implementations for double precision

targeting ASIC technology. The adders, multipliers, and converters for standard and HUB formats have been implemented using the Synopsys Design Compiler version Z-2007.03 and the TSMC 65-nm standard cell library, targeting clock frequencies ranging from 200 MHz to 1.1 GHz in 50-MHz steps. The area and power consumption of both approaches are compared for the same clock frequency.

Fig. 9 shows the area and power consumption of the standard and the HUB adders for single precision. In addition to the values in absolute terms, the HUB-to-standard ratio is also represented to facilitate the comparison. It can clearly be seen that the HUB adder always requires significantly less area than the standard adder, especially when the clock frequency increases. Specifically, the HUB adder requires between around 20% and 50% less area and power than the standard adder. On the other hand, the maximum frequency achievable for the standard adder is only 700 Mhz, whereas it is 800 MHz for the HUB adder. This means that the HUB adder is 14 % faster than the standard one. Furthermore, if we consider the fastest implementation in each case, besides being slower, the standard adder requires 63% more area and consumes 35% more power (even working at a lower speed) than the HUB adder. Therefore, the HUB approach simultaneously increases speed and reduces area and power consumption.

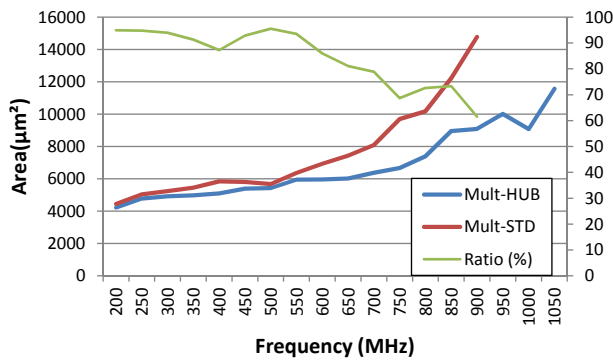
A similar comparison is provided for double precision numbers by Fig. 10. Again, an important area and power consumption reduction is achieved, although a little lower,

between 15% and 35%. Regarding the speed, the improvement is only 8% (650 MHz for HUB adder and 600MHz for standard one). However, taking the fastest implementation of both approaches, even being faster, the HUB adder requires 16% less area and consumes 13% less power.

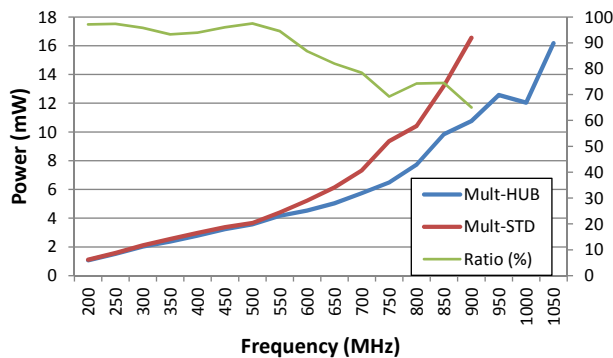
As expected, improvements in the multipliers are less than in the adders, although they are still very significant.

Fig. 11 shows the area required and the power consumption for both single-precision multipliers. The improvement is less than 10% for low frequencies, but becomes very significant when the frequency increases, achieving up to 38% and 35% reductions in area and power, respectively. Similar to adders, the HUB multiplier is 17% faster than the conventional one. If the fastest implementations of both approaches are compared, it can be seen that the HUB multiplier requires 22% less area and consumes slightly less power (2%). Similar behavior can be seen in the double precision implementation, as shown in Fig. 12. In this case, both the area and the power reduction increase 32% and the fastest implementation achieves a speed-up of 14%, with 12% less area and slightly more power consumption (2.6%).

Similarly, we also studied the circuits used to convert between single and double precision. Fig. 13(a) shows the area required for converting from double to single precision. It can be observed that the area of the HUB implementation is practically constant for the ranges of the frequencies tested. This is because the critical path is very short for this circuit. In this case, the elimination of the rounding logic dramatically

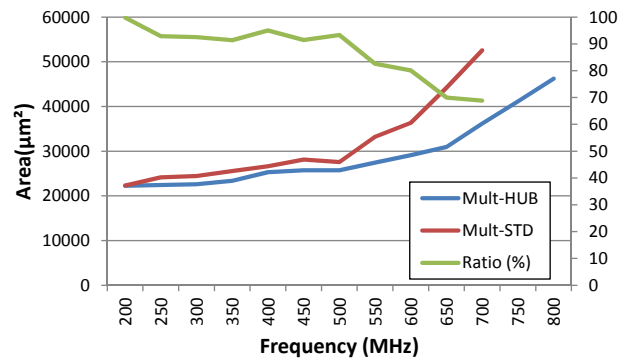


(a) Area

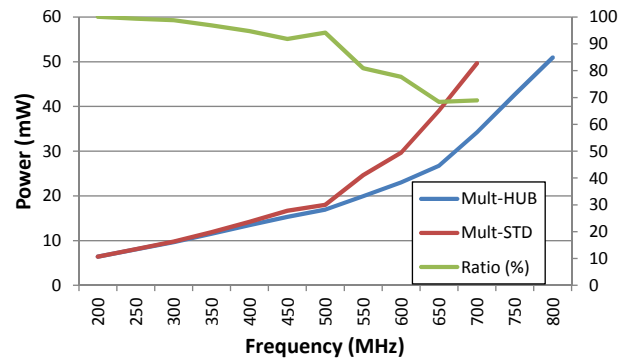


(b) Power

Fig. 11. Comparison of multiplier implementations for single precision



(a) Area



(b) Power

Fig. 12. Comparison of multiplier implementations for double precision

reduces the area. The circuit for the standard format requires between 2.5 and 4.4 as much area as the HUB approach. Regarding power consumption, although this reduction is slightly less, as shown in Fig. 13(b), it is still very high. The power consumption of the standard converter is between 2 and 3.6 times greater than the one for the HUB circuit. However, we should note that the relative impact of these circuits on the overall area or power is much lower than that on the adder or multiplier.

In contrast, the conversion from single to double precision is simpler in the standard approach. Similar to the previous HUB converter, the critical paths of these converters are too short to produce significant variations in the area at the frequencies tested. The standard converter requires $92.88\mu m^2$, whereas the HUB approach with tie-to-away requires $96.12\mu m^2$, which is about 3.5% more area. However, the HUB converter with tie-to-even-like rounding requires $116.64\mu m^2$, which is about 25 % more area. Accordingly, the tie-to-away HUB converter consumes only 1% more power than the standard converter, but the other converter requires 75% more power. Clearly the tie-to-away approach is the preferred solution for these kinds of conversions, unless this approach cannot be applied due to application restrictions. In any case, as stated, the area or power consumption of these circuits is much lower than arithmetic circuits, and thus their impact is very low.

In summary, the use of HUB FP numbers could significantly improve the implementation of arithmetic units. In all the tested cases, except for the converter from single to double

precision, HUB units clearly outperform standard units, in area, speed, and power consumption. Due to the nature of the improvement, which is based on the simplification of the FP algorithms themselves, it is expected that the adaptation of more advanced FP units to HUB numbers should result in obtaining more efficient circuits. However, we should recall that since the architectures used in this paper are very basic, the amount of improvement expected will be probably less when these more advanced architectures are adapted to HUB numbers. Therefore, the figures obtained in this study should be cautiously regarded as the upper bounds of the improvements that can be achieved.

V. CONCLUSION

This article presents a way to simplify FP systems through the use of HUB formats. When the preferred rounding mode is round-to-nearest, the implementation of the arithmetic unit for dealing with HUB FP numbers is much simpler than when using classic units. The substitution of classic formats for HUB formats is feasible, given that we have theoretically and experimentally demonstrated that both approaches have the same precision assuming the same storage requirements. Using basic architectures, we have shown how they could be adapted to support HUB numbers and also shown that the implementation figures are greatly improved by using HUB circuits. These results should be intended as a rough approximation of the improvement achievable for more advanced FP architectures.

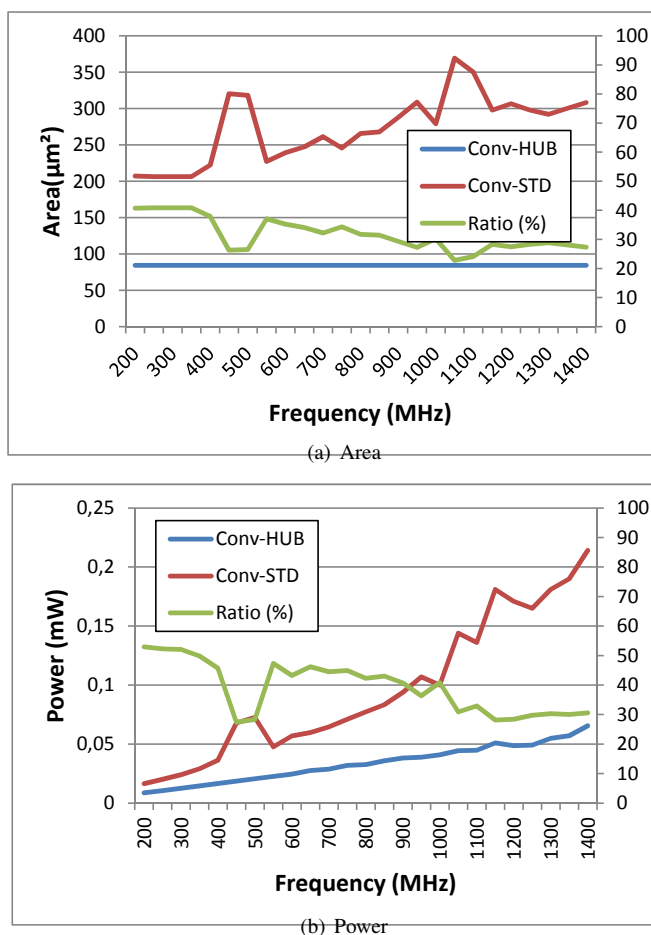


Fig. 13. Comparison of converter implementation from double to single precision

We should also note that several patent applications have been filed regarding several HUB circuits.

REFERENCES

- [1] M. D. Ercegovac and T. Lang, *Digital Arithmetic*. Morgan Kaufmann Publishers, 2003.
- [2] J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres, *Handbook of Floating-Point Arithmetic*. Birkhäuser Boston, 2010, ACM G.1.0; G.1.2; G.4; B.2.0; B.2.4; F.2.1., ISBN 978-0-8176-4704-9.
- [3] IEEE Task P754, *IEEE 754-2008, Standard for Floating-Point Arithmetic*. IEEE-STD, Aug. 2008. [Online]. Available: <http://ieeexplore.ieee.org/servlet/opac?punumber=4610933>
- [4] D. T. Matheny, D. V. Jaggard, and D. J. Seal, "Round increment in an adder circuit," US Patent 6 148 314, Nov. 2000.
- [5] D. R. Lutz and C. N. Hinds, "Data processing apparatus and method for performing floating point multiplication," US Patent 7 640 286, 12, 2009.
- [6] B. Boswell and K. Menezes, "Interface for performing parallel arithmetic and round operations," US Patent 6 055 555, 04, 2000.
- [7] G. Even and P.-M. Seidel, "A comparison of three rounding algorithms for IEEE floating-point multiplication," *Computers, IEEE Trans. on*, vol. 49, no. 7, pp. 638–650, Jul 2000.
- [8] N. Burgess, "Prenormalization rounding in IEEE floating-point operations using a flagged prefix adder," *Very Large Scale Integration (VLSI) Systems, IEEE Trans. on*, vol. 13, no. 2, pp. 266–277, Feb 2005.
- [9] N. Quach, N. Takagi, and M. Flynn, "Systematic IEEE rounding method for high-speed floating-point multipliers," *Very Large Scale Integration (VLSI) Systems, IEEE Trans. on*, vol. 12, no. 5, pp. 511–521, May 2004.

- [10] P. Kornerup, J.-M. Muller, and A. Panhaleux, "Performing arithmetic operations on round-to-nearest representations," *Computers, IEEE Trans. on*, vol. 60, no. 2, pp. 282–291, Feb 2011.
- [11] J. Hormigo and J. Villalba, "New formats for computing with real-numbers under round-to-nearest," *Computers, IEEE Trans. on*, vol. PP, no. 99, 2015.
- [12] —, "Optimizing DSP circuits by a new family of arithmetic operators," in *Signals, Systems and Computers, 2014 Asilomar Conference on*, Nov 2014, pp. 871–875.
- [13] S. D. Muñoz and J. Hormigo, "Improving fixed-point implementation of QR decomposition by rounding-to-nearest," in *Consumer Electronics (ISCE 2015), 19th IEEE Int. Symposium on*, June 2015, pp. 1–2.
- [14] J. Hormigo and J. Villalba, "Simplified floating-point units for high dynamic range image and video systems," in *Consumer Electronics (ISCE 2015), 19th IEEE Int. Symposium on*, June 2015, pp. 1–2.
- [15] S. Oberman, "Design issues in high performance floating point arithmetic units," *PhD thesis, Stanford University*, 1996.
- [16] G. Gerwig, H. Wetter, E. Schwarz, C. Krygowski, B. Fleischer, and M. Kroener, "The IBM eServer z990 floating-point unit," *IBM J. of Research and Development*, vol. 48, no. 3-4, pp. 311–322, 2004.
- [17] P.-M. Seidel and G. Even, "Delay-optimized implementation of IEEE floating-point addition," *Computers, IEEE Trans. on*, vol. 53, no. 2, pp. 97–113, Feb 2004.
- [18] M. Jaiswal, R. Cheung, M. Balakrishnan, and K. Paul, "Unified architecture for double/two-parallel single precision floating point adder," *Circuits and Systems II: Express Briefs, IEEE Trans. on*, vol. 61, no. 7, pp. 521–525, July 2014.
- [19] J. Sohn and E. Swartzlander, "Improved architectures for a fused floating-point add-subtract unit," *Circuits and Systems I: Regular Papers, IEEE Trans. on*, vol. 59, no. 10, pp. 2285–2291, Oct 2012.
- [20] E. Quinnell, E. Swartzlander, and C. Lemonds, "Bridge floating-point fused multiply-add design," *Very Large Scale Integration (VLSI) Systems, IEEE Trans. on*, vol. 16, no. 12, pp. 1727–1731, Dec 2008.
- [21] S.-R. Kuang, J.-P. Wang, and H.-Y. Hong, "Variable-latency floating-point multipliers for low-power applications," *Very Large Scale Integration (VLSI) Systems, IEEE Trans. on*, vol. 18, no. 10, pp. 1493–1497, Oct 2010.
- [22] J. Tong, D. Nagle, and R. Rutenbar, "Reducing power by optimizing the necessary precision/range of floating-point arithmetic," *Very Large Scale Integration (VLSI) Systems, IEEE Trans. on*, vol. 8, no. 3, pp. 273–286, June 2000.
- [23] H. Suzuki, H. Morinaka, H. Makino, Y. Nakase, K. Mashiko, and T. Sumi, "Leading-zero anticipatory logic for high-speed floating point addition," *Solid-State Circuits, IEEE Journal of*, vol. 31, no. 8, pp. 1157–1164, Aug 1996.

PLACE
PHOTO
HERE

Javier Hormigo received a M.Sc and a Ph.D., both in Telecommunication Engineering, from the Universidad de Malaga, Spain, in 1996 and 2000, respectively. He was a member of the Image and Vision Department of the Instituto de Optica, Madrid, Spain, in 1996. He joined the Universidad de Malaga in 1997 and is currently Associate Professor in the Computer Architecture Department. His research interests include computer arithmetic, specific application architectures, and FPGA.

PLACE
PHOTO
HERE

Julio Villalba received a B.Sc degree in Physics in 1986 (Universidad de Granada, Spain) and a Ph.D in Computer Engineering in 1995 (Universidad de Malaga, Spain). During 1986-1991, he worked in the R&D Department of Fujitsu Spain and was an assistant professor. Since 2007, he has been a Full Professor in the Department of Computer Architecture at the Universidad de Malaga. Currently he is an Associate Editor of IEEE Trans. on Emerging Topics in Computing and a steering committee member of ARITH23. His research interests include computer arithmetic and specific application architectures.